Improving data transfer rate of Hadoop MapReduce framework using data blocks for massive data

[1]Sujit Roy, [2]Md. Humaun Kabir, [3]Ripan Roy, [4]Md. Zahidul Alam
[1]Lecturer, Department of Computer Science and Engineering, [2]Lecturer, Department of Computer Science and Engineering, [3]Lecturer, Department of Mathematics, [4]Lecturer
Bangamata Sheikh Fojilatunnesa Mujib Science & Technology University, Jamalpur, Bangladesh

Abstract - **In this research paper, a new technique has been proposed to process the massive data in Hadoop MapReduce framework to improve data rate by using synchronous data transmission, sending block of data from source to destination. The proposed method shows how to divide the data blocks in an efficient manner for achieving satisfactory data transfer rate by adjusting the split size or using appropriate size of staffs. In traditional system, normally data transfer is accomplished through a small block of 8 bit while in the proposed system data transfer is performed through a block size of 80 byte to 132 byte. Moreover, the traditional system needs to add 3 extra bits with a block of data during data transmission while the proposed system attaches additional 32 byte with a block of data. For this reason, our proposed system takes more time to transfer small size data but it transfer big size data very faster than the current systems. From the simulation results, it is observed that the proposed model is more efficient and provides satisfactory performance for the big size data.**

keywords - **MapReduce, Massive Data, Incremental Processing, Hadoop, Distributed Computing, HDFS**

I. Introduction

Data processing is the major challenge in big data because it contains different types of data, and computations cannot be accomplished by the estimated database and data mining techniques. An investigation study states that big data contents are produced increasingly day by day. IBM states that 2.5 billion gigabytes of data are produced in a single day. In 2012, in a single dataset, the size of the data likely to rise from few terabytes (TB) to many petabytes (PB) and about 90% of the whole data in the world today is produced in the last two years only. In this era of data science, in its entirety, the term "data" is further redefined, which is popularly known as "Big Data" today. The concept of Big Data is relatively new which has more than a few characteristics and has a large scope of further research. The Big Data is termed to have five concerns are volume of the data, velocity of the data, variety of the data, veracity of the data, and the value of the data.

Hadoop is an open source framework which provides a consistent storing of large data collections over multiple service servers and parallel processing of data analysis. MapReduce is a programming model and an accompanying employment for processing and generating big data sets with a parallel distributed algorithm on a Hadoop cluster. The MapReduce framework was announced by Google in 2004 [7]. Hadoop MapReduce is a software framework built on top of Hadoop used for processing large data collections in parallel on Hadoop clusters [1]. The algorithm of MapReduce is based on a common map and reduce programming model widely used in functional programming. It is particularly suitable for parallel processing as each map or reduce task operates independent of one another. MapReduce jobs are mostly I/O bound as 70% of a single job is found to be I/O-intensive tasks [2]. A characteristic MapReduce job is divided into three sequential I/O-bound phases:

1. Map phase: Locations of input data blocks distributed over multiple data nodes are retrieved via NameNode. Blocks are loaded into memory from local disk and each map task processes corresponding blocks. Intermediate results from each map task are materialized in map output buffers.
2. Shuffle phase: Once a map task is completed, spilled contents are merged and shuffled across the network to corresponding reduce tasks.
3. Reduce phase: Each reduce task process received key groups. Similar to the map phase, reduce inputs are temporarily stored in reducer output buffers and periodically spilled to disks.

II. Hadoop Framework

Hadoop is an open source platform which is used effectively to handle the big data applications. The two core concepts of the Hadoop are Map reduce and Hadoop distributed file system (HDFS). HDFS is the storage machine and map reduce is the programming language. Results are manufactured faster than other Traditional database operations. Pig and Hive are the two languages which help us to program the MapReduce framework within short period of time . Hadoop contains the distributed file system in order to handle the large range of data. Major features of Hadoop are reliability, data locality, cost effectiveness, efficient computation, high data locality and faster data processing. Reliable, stable and consistent data is generated, which means data contents will be the same all the time after processing set of inputs. Hadoop has several different vendors: Cloudera, Horton works, MapR, Amazon Elastic MapReduce, IBM Info sphere big Insights are some of them.

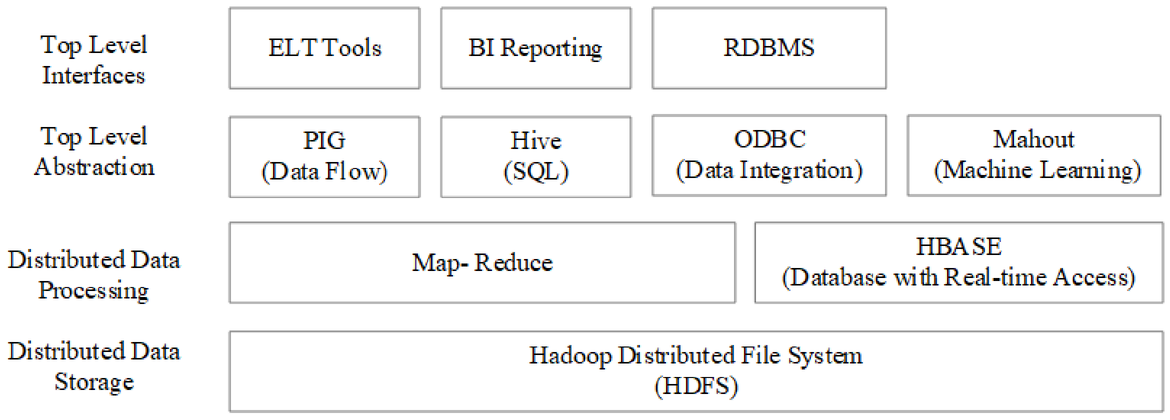

Figure 1: Architecture of Hadoop.

III. Hadoop Distributed File System

HDFS was designed in the project NUTCH. Input data are split into different portions and stored in HDFS. The default portion size is 64MB. HDFS has blocked oriented architecture. Each block has fixed size and is stored in the Hadoop cluster. These different blocks are called as data node and they contains the actual data. The data nodes are stored in different machines at different clusters. Since the data is processed in the same cluster where it is stored, it avoids the problems related to transferring of data from one place to another. Thus the HDFS provide reliable and fast access to the stored data. Name node stores the metadata for the file system across each Hadoop cluster. Name node is stored in the main memory, so it allows fast random access. The data stored in the name node are persistent and due to this failure will cause the permanent loss of the data. Because its contain all the links to the data nodes. To avoid the loss of information, the secondary name node is maintained. It contains the image of the name node and the edit logs. When failure comes, based on these log details the data can be retrieved. The secondary name node cannot be replaced directly instead of the actual name node [4] [7].

IV. Map Reduce Logical View

The Map Reduce framework helps easily to write applications which process large amounts of data, up to several Terabytes, in parallel on large clusters, consisting of hundreds of nodes made using commodity hardware, with highly reliable and fault-tolerant manner. The key idea behind MapReduce is mapping the data set into a collection of <key, value> pairs, and then reducing all the pairs with the same key. Map takes one pair of data with a type in one data domain, and returns a list of pairs in a different domain:

$$\mathrm{Map}(k1, v1) \rightarrow \mathrm{List}(k2, v2)$$

The Map function is applied in parallel to every pair (keyed by κ1) in the input dataset. This produces a list of pairs (keyed by κ2) for each call. After that, the MapReduce framework collects all pairs with the same key (κ2) from all lists and groups them together, creating one group for each key. The Reduce function is then applied in parallel to each group, which in turn produces a collection of values in the same domain:

$$\mathrm{Reduce}(k2, \mathrm{List}(v2)) \rightarrow \mathrm{List}(v3)$$

Each Reduce call typically produces either one value v3 or an empty return, though one call is allowed to return more than one value. A MapReduce task separates the input data into several independent blocks each of which are then processed by the map process in parallel as shown in Figure 2. The output of the Map task is then passed to the shuffle task. The Reducer stage has three phrases: shuffle phase, sort phase, and the reduce phase. In the Shuffle phase, the output from the Mappers is given to the Reducer after sorting them. The MapReduce framework then gathers the portion of the output from all the mappers using HTTP protocol. Through the sorting phase, the MapReduce programming framework then groups the output that have the same input by the assigned key value. Both the shuffle phase and sort phase happens parallelly. Several output pieces from the mapper program are fetched simultaneously as they get merged. It is often used to track how intermediate keys are grouped together and be used additionally to simulate secondary sort on values. The final phase of the MapReduce program is the reduce phase. The reduce phase uses results from shuffler as input and run them across several reducers at the same time. The reduce tasks are then consolidated into the final result.

The advantages of MapReduce programming are scalability, cost-effective solution, flexibility in environment, fast execution, security and authentication, parallel processing, availability and resilient nature. The programming processing divides the tasks in a manner that allows the execution of the independent task in parallel. Hence this parallel processing makes it easier for the processes to take on each of the tasks which help to run the program in much less time. Map reduce has a large capability when it comes to large data processing compared to traditional RDBMS systems.

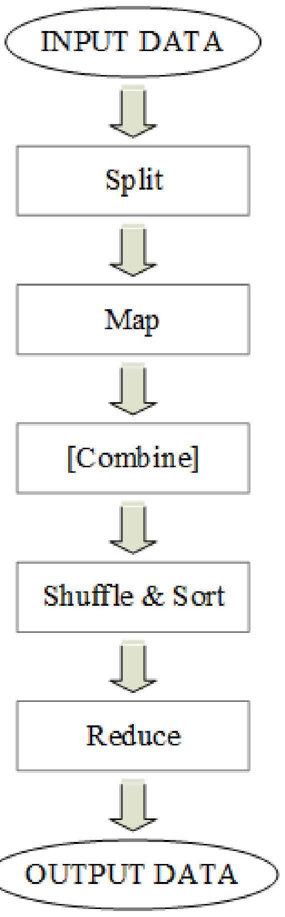

Figure 2: Block diagram of MapReduce data flow.

V. Related Work

Hadoop MapReduce is an open-sourced framework that supports MapReduce programming model introduced by Google [2]. The MapReduce model consists of two primitive functions: Map() and Reduce(). Users can define Map() and Reduce() functions. Each processing job in Hadoop is broken down to as many Map tasks as input data blocks and one or more Reduce tasks.

Hadoop MapReduce also utilizes HDFS as an underlying storage layer [11]. HDFS is a block-structured file system that supports fault tolerance by data partitioning and block replication, managed by a single or two master nodes. Some approaches for improving the I/O performance in MapReduce-based programs have been proposed. Readers are referred to a recent survey for MapReduce and its improvements [3]. In this survey paper [18], we are providing a state of the art overview of Cloud-centric Big Data placement together with the data storage methodologies. It is an attempt to highlight the actual correlation between these two in terms of better supporting Big Data management. Clustering algorithms have emerged as an alternative powerful meta-learning tool to analyze accurately the massive volume of data generated by modern applications. In particular [22], their main goal is to categorize data into clusters such that objects are grouped in the same cluster when they are similar according to specific metrics.

Hive [6] is an open-source project, which aims at providing a data warehouse solution on the Hadoop framework. It supports ad hoc queries with an SQL-like query language. Hive evaluates its SQL-like query by compiling the query into a directed acyclic graph that is composed of multiple MapReduce jobs. Hive also maintains a system catalog that provides schema information and data statistics, similar to other relational database systems. HBase [7] is an open-source Java implementation of Google's Big table [8]. HBase is a wide-column store, which maps two arbitrary string values (row key and column key) and timestamp into an associated arbitrary byte array, working on HDFS. It features data compression, the use of bloom filter for checking the existence of data, and a log-structured storage. HBase is not a relational database, rather known to be a sparse, distributed multisorted map which works better for treating sparse data such as web addresses.

Replica placement policy for HDFS is subject of several works (Dai, 2016), (Mansouri, 2016), (Park, 2016). The distribution of replica across cluster nodes allows to improve the performance of HDFS by balancing the simultaneous access to a data. The full storage size limits the usage of replica in suggested replica placement strategies. But all of proposed policies show better performance in case of reading of a data in compare with standard Hadoop replica allocation. The performance of each data node in a heterogeneous Hadoop cluster differs, and the number of slots that can be numbered to simultaneously execute tasks differs. For this reason, Hadoop is susceptible to replica placement problems and data replication problems [15].

VI. Modeling of Hadoop Framework

The Hadoop framework is used to process the big data applications. It joins multiple datasets together. There are different step in the modeling of the Hadoop framework. At first, the contents are stored into the HDFS and then we can process the data using the mapreduce concept. HDFS splits the contents into different chunks and save in different data nodes of the Hadoop cluster. After that mapreduce algorithm will map the contents from different data nodes as the key value pair and finally reducer will process the contents to get the meaningful data. The block diagram of the Hadoop framework is shown in Figure 3.

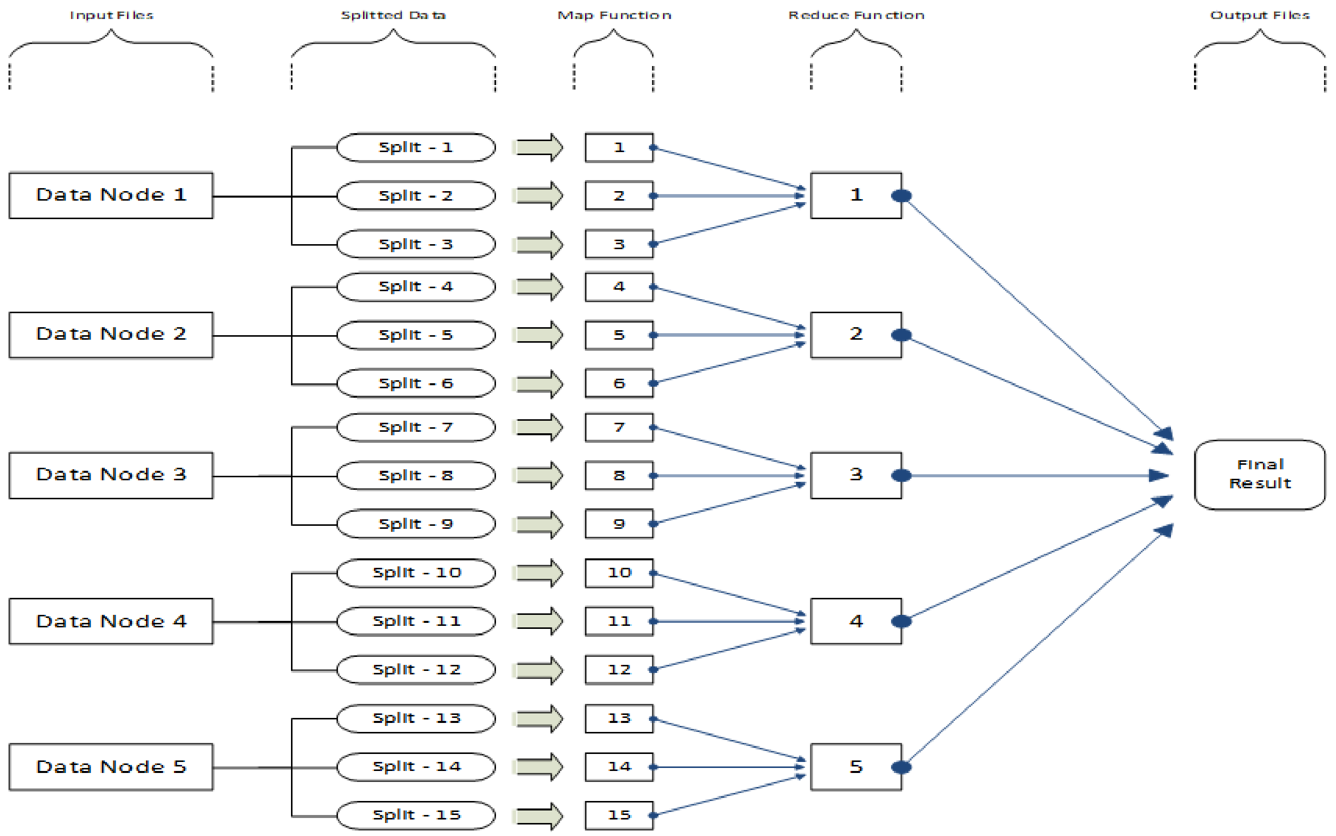

Figure 3: MapReduce accomplishment flow.

VII. Proposed Methods

A. Proposed Data Transmission Methodology

In this proposed method, the data blocks are divided to get the maximum data transmission efficiency. The target data are stored in a primary storage device and then transmitted the data, the format of a data container shown in Figure 4 will take 80-132 characters in size. Trailer and header are also connected to the data field. Both the trailer and header contain an 8 bits flag and control field. The data is divided into blocks (packets or frames) and transmit one block each time. There is block interval between two blocks. Every block data contain 2 bytes header information at the front and 2 bytes trailer information at the end.

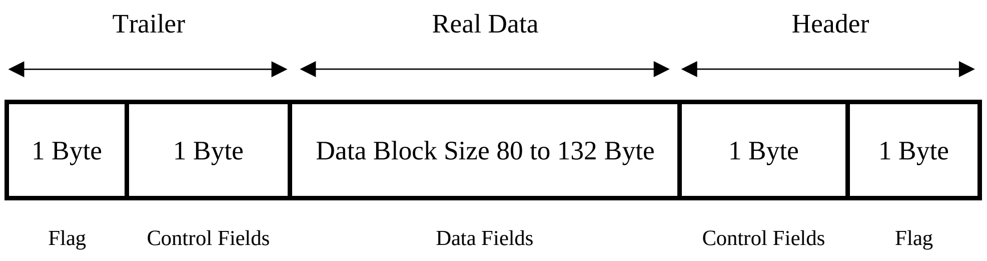

Figure 4: The layout of data transmission block.

B. Algorithm of Proposed Approach

Step1: The input reader reads data from stable storage (typically a distributed file system) and generates key/ value pairs.
Step2: Split the input file (0………………………..N)
Step3: The Map () is used for the split file (split 0 ………………..split N)
Step4: Then the split files are shuffling according to their file category.
Step5: Apply Reduce () to optimize it. The Reduce can iterate through the values that are associated with that key and produce zero or more outputs.
Step6: Analyze the files efficiency by using the formula.

$$\text{Data Transmission Efficiency} = \frac{\text{Real Data}}{\text{Total Data}} \times 100\%$$

Here, Total Data = Real Data + Overhead Data

Step7: Finally, we can analyze the data transfer rate by using the formula

$$\text{Data Transfer Rate(Kbps)} = \frac{\text{Total File Size (MB)}}{\text{Total Transfer Time (Sec)}}$$

C. Mathematical Analysis

Now we assume the Throughput of the data file.

Throughput is the ratio of the file size and time.

$$\text{Data Transfer Rate (N)} = \frac{\sum_{i=0}^{N} \text{File Size}_i}{\sum_{i=0}^{N} \text{Time}_i} \text{ Mb/ Sec [File Size (MB)/Transfer Speed (kbps) = Time(s)]}$$

$$\text{Average IO Rate (N)} = \frac{\sum_{i-0}^{N} \text{Rate}_i}{N}$$

To analyze the efficiency by using the formula

$$\text{Data Efficiency } (\mu) = \frac{R_d}{T_d} \times 100\%$$

$$\text{Total Data}, T_d = R_d + O_d$$

$$\text{Overhead Data}, O_d = \left|\frac{\text{Real Data}}{\text{Block size}}\right| \times 32$$

Here, T_d = Total Data, R_d = Real Data, O_d = Overhead Data

$$\text{Data Transfer Speed (kbps)} = \frac{\text{File Size (MB)}}{\text{Time (s)}}$$

VIII. Result and Discussion

In this research, various sizes of data are used to process in Traditional and proposed methods to get a comparison between the systems. The outcomes of the proposed system showed a better data transfer average efficiency than the Traditional methods. When the size of the primary storage range is 5 to 60 MB then the Traditional method carry out the better data transfer average efficiency than the proposed method which is shown in the following Table 1. In the same table, if the size of the file range is 65 to 100 MB, then the data transfer average efficiency of Traditional method is approximately equivalent to proposed method. The data transfer average efficiency of proposed method is greater than the Traditional method for 100 MB above the file size. We can find the data rate of Traditional and proposed methods to get a comparison between the systems from the table 2. The outcomes of the proposed system showed a better data rate than the Traditional methods for immense file size. The Traditional method average data rate o.72 bpns and proposed method average data rate 0.97bpns.

Primary Storage Space (MB)	Data Transfer Average Efficiency (%)	
	Traditional Method	Proposed Method
5 to 60	69.3335	46.2833
65 to 100	71.8255	71.7375
100 to above	72.7152	94.8848

Table 1: Different Primary Storage file size wise average efficiency.

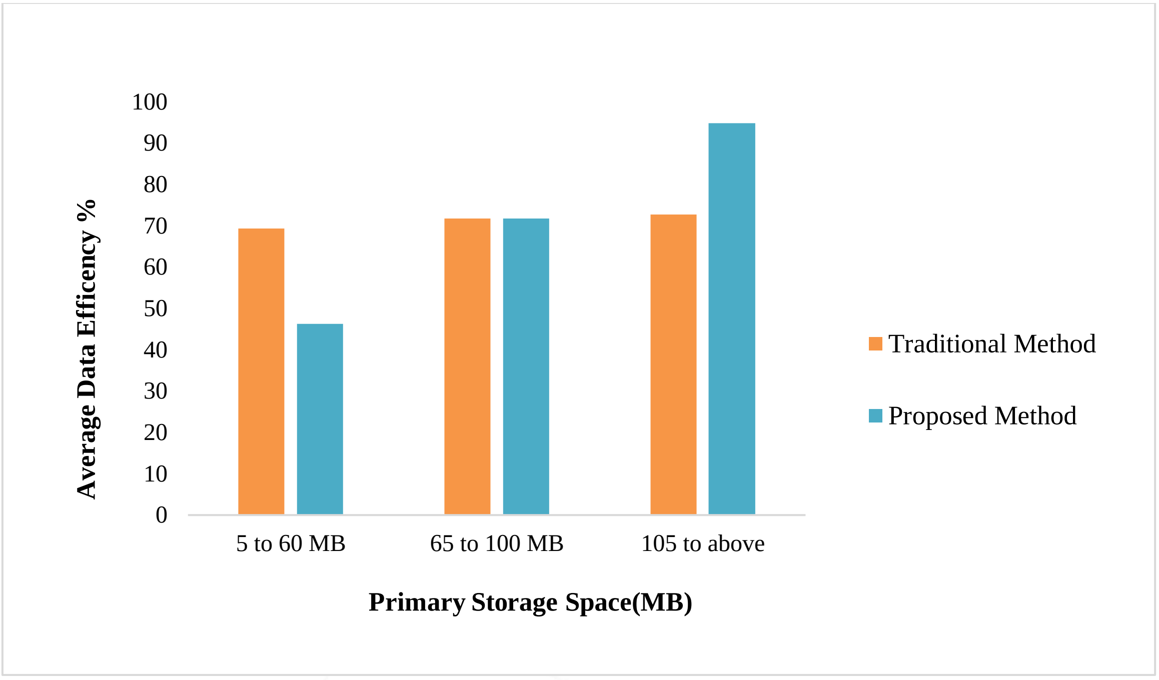

Figure 5: Different Primary Storage file size wise average efficiency.

File Size	Traditional Method			Proposed Method		
	Block	**Total Time (sec)**	**Bandwidth (bpns)**	**Block**	**Total Time (ns)**	**Bandwidth (bpns)**
1 Byte	1	11	0.727272727	1	40	0.20000
100 Byte	100	1100.099	0.727207279	1	832	0.961538
500 Byte	500	5500.499	0.72720675	4	4128.03	0.968985
1 KB	1024	11265.023	0.727206682	8	8448.07	0.969689
100 KB	102400	1126502.399	0.727206618	776	844039.8	0.970578
500 KB	512000	5632511.999	0.727206618	3879	4220167	0.970578
1 MB	1048576	11535384.58	0.727206618	7944	8642895	0.970578
100 MB	104857600	1153538458	0.727206618	794376	8.64E+08	0.970579
500 MB	524288000	5767692288	0.727206618	3971879	4.32E+09	0.970579
1 GB	1073741824	11812233806	0.727206618	8134408	8.85E+09	0.970579
100 GB	1.07374E+11	1.18122E+12	0.727206618	8.13E+08	8.85E+11	0.970579
500 GB	5.36871E+11	5.90612E+12	0.727206618	4.07E+09	4.43E+12	0.970579
1 TB	1.09951E+12	1.20957E+13	0.727206618	8.33E+09	9.06E+12	0.970579
100 TB	1.09951E+14	1.20957E+15	0.727206618	8.33E+11	9.06E+14	0.970579
500 TB	5.49756E+14	6.04786E+15	0.727206618	4.16E+12	4.53E+15	0.970579

Table 2: Comparison of Data Transfer Rate between Traditional and Proposed Method.

IX. Conclusion and Future Work

Map Reduce is an effective programming model for significant data-intensive multiplying applications. Hadoop, an open source implementation of Map Reduce, has been widely used. The communication overhead from the big data sets' transmission affects the performance of Hadoop greatly. Hadoop is comprehensively used for considered decision making in the big data applications. It has many application areas like sophisticated data mining, pattern recognition, content optimizing, marketing analysis, network analysis, large data transformations, text processing etc. Hadoop framework can be used to make informed decisions in logistic freight in order to perform the freight audit.

This study focuses on to measure the better data efficiency of periodic tasks that run on HADOOP platforms. It also studied how to divide the data blocks efficiently to distribute and manage the data. Data block distribution according to the proposed architecture in HADOOP achieved around 22% more efficiency than the other Traditional systems.

Next we plan to work on Bandwidth in HADOOP clustering. As the HADOOP contains different categories of file extension, so it would be possible to increase data transmission rate by grouping files before execution. Data block distribution according to the proposed method in HADOOP achieved the data transfer rate better than the other traditional methods.

References

[1] Yongwei Wu; Feng Ye; Kang Chen; Weimin Zheng, "Modeling of Distributed File Systems for Practical Performance Analysis," IEEE Transactions on Parallel and Distributed Systems, vol.25, no.1, pp.156- 166, Jan.2014, doi:10.1109/TPDS.2013.19.

[2] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.

[3] K. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, “Parallel data processing with MapReduce: a survey,” ACM SIGMOD Record, vol. 40, no. 4, pp. 11–20, 2012.

[4] Y. He, R. Lee, Y. Huai et al., “RCFile: a fast and space efficient data placement structure in mapreduce-based warehouse systems,” in Proceedings of 27th IEEE ICDE Conference, pp. 1199–1208, Hannover, Germany, April 2011.

[5] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian, “A comparison of join algorithms for log processing in MapReduce,” in Proceedings of ACM SIGMOD Conference, pp. 975–986, Indianapolis, IN, USA, June 2010.

[6] A. Thusoo, J. S. Sarma, N. Jain et al., “Hive: a warehousing solution over a map-reduce framework,” Proceedings of VLDB Endowment, vol. 2, no. 2, pp. 1626–1629, 2009.

[7] L. George, HBase: the Definitive Guide: Random Access to your Planet-Size Data, O’Reilly Media Inc., Sebastopol, CA, USA, 2011.

[8] F. Chang, J. Dean, S. Ghemawat et al., “Bigtable: a distributed storage system for structured data,” ACM Transactions on Computer Systems (TOCS), vol. 26, no. 2, pp. 1–26, 2008.

[9] A. Pavlo, “A comparison of approaches to large-scale data analysis,” in Proceedings of SIGMOD Conference, pp. 165–178, Providence, RI, USA, June 2009.

[10] M. Zaharia, M. J. Franklin, A. Ghodsi et al., “Apache Spark: a unified engine for big data processing,” Communications of ACM (CACM), vol. 59, no. 11, pp. 56–65, 2016.

[11] K. Shvachko, “The Hadoop distributed file system,” in Proceedings of IEEE Symposium on Mass Storage Systems and Technologies (MSST), pp. 1–10, Lake Tahoe, NV, USA, May 2010.

[12] Y. Huai, X. Zhang, A. Chauhan et al., “Major technical advancements in Apache Hive,” in Proceedings of ACM SIGMOD Conference, pp. 1235–1246, Snowbird, UT, USA, June 2014.

[13] W. Ligon, III and R. Ross, “Implementation and Performance of a Parallel File System for High performance Distributed Applications,” Proc, IEEE Fifty Int’l Symp. High Performance Distributed Computing,pp471-480, 1996.

[14] Zhu, Nan; Liu, Xue; Liu, Jie; Hua, Yu, "Towards a cost-efficient MapReduce: Mitigating power peaks for Hadoop clusters," Tsinghua Science and Technology, vol.19, no.1, pp.24,32, Feb. 2014 doi: 10.1109/TST.2014.6733205.

[15] Park, D. K. An efficient Hadoop data replication method design for heterogeneous clusters. In Proceedings of the 31st Annual ACM Symposium on Applied Computing (pp. 2182-2184). ACM, 2016.

[16] Sohangir S, Wang D, Pomeranets A, Khoshgoftaar TM. Big data: deep learning for financial sentiment analysis. J Big Data. 2018; 5(1):3. doi: 10.1186/s40537-017-0111-6.

[17] Apache Hadoop. http://hadoop.apache.org (2018). Accessed 27 June 2019.

[18] Mazumdar S, Seybold D, Kritikos K, Verginadis Y. A survey on data storage and placement methodologies for cloud-big data ecosystem.J Big Data. 2019; 6(1):15. doi:10.1186/s40537-019-0178-3.

[19] Oussous A, Benjelloun FZ, Lahcen AA, Belfkih S. Big data technologies: a survey. J King Saud Univ Comput Inf Sci. 2017;30 (4):431–48.

[20] Fang H et al. A survey of big data research. In: IEEE network; 2015. p. 6–9.

[21] Fahad A, et al. A survey of clustering algorithms for big data: taxonomy and empirical analysis. IEEE Trans Emerg Top Comput. 2014;2(3):267–79.